

---

# 1.1 RETI INFORMATICHE

## CORSO DI LAUREA IN INGEGNERIA INFORMATICA

### SPECIFICHE DI PROGETTO A.A. 2012/2013

Il progetto consiste nello sviluppo di un'applicazione client/server. Sia il server che il client dovranno essere mono-processo e sfrutteranno l'I/O multiplexing per gestire più input ed output simultaneamente. L'applicazione da sviluppare è il gioco del TRIS seguendo il paradigma peer2peer. La mappa del gioco è la seguente:

7	8	9
4	5	6
1	2	3

Ogni casella è identificata dal numero riportato in figura.

Durante una partita ogni giocatore ha a disposizione uno dei due simboli di gioco: 'X' oppure 'O'.

Il server avrà il compito di memorizzare gli utenti connessi e le porte su cui rimarranno in ascolto.

Lo scambio di informazioni tra client e server avverrà tramite socket TCP. Queste informazioni saranno solo informazioni di controllo che serviranno per implementare la comunicazione peer2peer. Lo scambio di messaggi tra i client avverrà tramite socket UDP.

#### 1.1.1 Lato client

Il client deve essere avviato con la seguente sintassi:

```
./tris_client <host remoto> <porta>
```

dove:

- <host remoto> è l'indirizzo dell'host su cui è in esecuzione il server;
- <porta> è la porta su cui il server è in ascolto.

I comandi disponibili per l'utente devono essere:

- !help
- !who
- !quit
- !connect nome\_utente
- !disconnect
- !show\_map

• !hit num\_cell

Il client deve stampare tutti gli eventuali errori che si possono verificare durante l'esecuzione.

All'avvio della connessione il client **deve** inserire il suo username e la porta di ascolto UDP per i comandi relativi al gioco.

Un esempio di esecuzione è il seguente:

```
$ ./tris_client 127.0.0.1 1234

Connessione al server 127.0.0.1 (porta 1234) effettuata con successo

Sono disponibili i seguenti comandi:
* !help --> mostra l'elenco dei comandi disponibili
* !who --> mostra l'elenco dei client connessi al server
* !connect nome_client --> avvia una partita con l'utente nome_client
* !disconnect -->disconnette il client dall'attuale partita intrapresa
con un altro peer
* !quit --> disconnette il client dal server
* !show_map --> mostra la mappa di gioco
* !hit num_cell --> marca la casella num_cell (valido solo quando è
il proprio turno)

Inserisci il tuo nome: client1
Inserisci la porta UDP di ascolto: 1025
```

## Implementazione dei comandi

help: mostra l'elenco dei comandi disponibili.

Esempio di esecuzione:

```
Sono disponibili i seguenti comandi:
* !help --> mostra l'elenco dei comandi disponibili
* !who --> mostra l'elenco dei client connessi al server
* !connect nome_client --> avvia una partita con l'utente nome_client
* !disconnect -->disconnette il client dall'attuale partita intrapresa
con un altro peer
* !quit --> disconnette il client dal server
```

```
* !show_map --> mostra la mappa di gioco
* !hit num_cell --> marca la casella num_cell (valido solo quando è
il proprio turno)
```

who: mostra l'elenco dei client connessi (OPZIONALE: mostrare anche lo stato del client, cioè dire se è occupato o meno in una partita).

Il server mantiene una lista dei client connessi in ogni momento. La lista contiene gli username, l'indirizzo ip e la porta di ascolto UDP con cui i client si sono registrati al momento della connessione.

Esempio di esecuzione:

```
> who
Client connessi al server: client1 client2 client3 client4
>
```

!connect nome\_client: il client avvia una partita con l'utente nome\_client.

Un client vuole avviare una partita con un altro client di nome nome\_client.

Gli errori da gestire sono:

- nome\_client inesistente,
- errori a livello protocollare.
- nome\_client già occupato in una partita

Più in dettaglio il client farà richiesta al server (sempre tramite tcp) per sapere se esiste l'utente nome\_client.

Se esiste e non è occupato il server manderà una richiesta al client nome\_client per sapere se è intenzionato ad accettare la partita con il client. Se la risposta è affermativa allora il server comunicherà al client l'indirizzo ip e porta di ascolto UDP del client nome\_client. Se negativa il server risponderà con uno specifico messaggio di errore. Il simbolo 'X' è assegnato all'utente che ha richiesto la connessione.

La concorrenza tra standard input e socket dovrà essere gestita sempre tramite select.

Esempio di esecuzione:

```
> !connect nome_client
nome_client ha accettato la partita
partita avviata con nome_client
il tuo simbolo è: X
E' il tuo turno:
#!hit 3
E' il turno di nome_client
nome_client ha marcato la casella 7
#!hit 5
...
...
```

---

Possibile segnalazione di errore:

```
> !connect nome_client
Impossibile connettersi a nome_client: utente inesistente.
```

```
> !connect nome_client
Impossibile connettersi a nome_client: l'utente ha rifiutato la partita.
```

Partita Avviata: Quando la partita è avviata il sistema dovrà accettare i seguenti comandi:

- 1) !disconnect
- 2) !quit
- 3) [opzionale] !who
- 4) [opzionale] !help
- 5) !hit num\_cell
- 6) !show\_map

Se una partita è avviata si deve capire dal primo carattere della shell:

1 > shell comandi (sono accettati solo i comandi di base, se immesso altro viene restituito un errore)

2 # shell partita (si accettano i comandi relativi al gioco)

!disconnect: disconnette il client dall'attuale partita.

```
# !disconnect
Disconnessione avvenuta con successo: TI SEI ARRESO
```

Quando un client esegue una disconnessione comunicherà

1 al server (tramite tcp) che l'utente è di nuovo libero

2 all'altro client (tramite udp) che è stata effettuata una disconnessione

Il client che riceve il messaggio di disconnessione dovrà comunicare al server che è di nuovo libero e stampare a video un messaggio di vittoria.

!quit: il client chiude il socket con il server, il socket udp ed esce. Il server stampa un messaggio che documenta la disconnessione del client. Il server, inoltre, dovrà gestire in maniera appropriata la disconnessione di un cliente.

---

```
> !quit  
Client disconnesso correttamente
```

!show\_map: il client mostra la mappa di gioco, in particolare per ogni casella dovrà essere mostrato lo stato.

### 1.1.1 Lato server

Il programma **tris\_server** si occupa di gestire le richieste provenienti dai client. Il server **tris\_server** tramite l'uso della select, accetterà nuove connessioni tcp, registrerà nuovi utenti e gestirà le richieste dei vari client per aprire nuove partite.

La sintassi del comando è la seguente:

```
./tris_server <host> <porta>
```

dove:

- <host> è l'indirizzo su cui il server viene eseguito;
- <porta> è la porta su cui il server è in ascolto.

Una volta eseguito, **tris\_server** deve stampare a video delle informazioni descrittive sullo stato del server (creazione del socket di ascolto, connessioni accettate, operazioni richieste dai client ecc.).

Un esempio di esecuzione del server è il seguente:

```
$ ./tris_server 127.0.0.1 1235  
Indirizzo: 127.0.0.1 (Porta: 1235)  
Connessione stabilita con il client  
pippo si e' connesso  
pippo è libero  
Connessione stabilita con il client  
pluto si e' connesso  
pluto è libero  
pippo si è connesso a pluto  
pluto si è disconnesso da pippo  
pippo è libero  
pluto è libero
```

### 1.1.1 Avvertenze e suggerimenti

- Modalità di trasferimento dati tra client e server (e viceversa).

---

1) Client e server si scambiano dei dati tramite socket TCP. Prima che inizi ogni scambio è necessario che il ricevente sappia quanti byte deve leggere dal socket. **NON È AMMESSO CHE VENGA INVIATI SU SOCKET NUMERI ARBITRARI DI BYTE.**

• **Il client si disconnette in automatico da una eventuale partita dopo 1 minuto di inattività:**

1 Non viene scritto niente nello standard input per un minuto

2 Non si riceve niente sul socket udp per un minuto

### 1.1.1

### 1.1.2 Valutazione del progetto

Il progetto viene valutato durante lo svolgimento dell'esame. Il tutto verrà fatto girare su una macchina con sistema operativo debian. Si consiglia caldamente di testare il sorgente su una macchina debian prima di venire all'esame. La valutazione prevede le seguenti fasi.

**1.Compilazione dei sorgenti.** Il client e il server vengono compilati attivando l'opzione -Wall che abilita la segnalazione di tutti i warning. Si consiglia vivamente di usare tale opzione anche durante lo sviluppo del progetto, *interpretando i messaggi forniti dal compilatore.*

**2.Esecuzione dell'applicazione.** Il client e il server vengono eseguiti simulando una tipica sessione di utilizzo. In questa fase si verifica il corretto funzionamento dell'applicazione e il rispetto delle specifiche fornite.

**3.Esame del codice sorgente.** Il codice sorgente di client e server viene esaminato per controllarne l'implementazione.